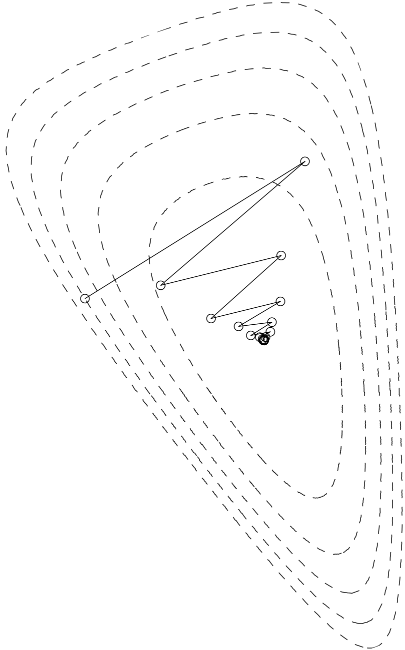


Optimization Algorithms

Approximate Newton Methods

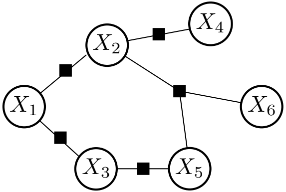
Gauss-Newton, BFGS, conjugate gradient

Marc Toussaint
Technical University of Berlin
Winter 2024/25

Approximate Newton Methods

- In high dimensions, computing exact Newton steps can be inefficient:
  - Computing and storing the dense Hessian $H \in \mathbb{R}^{n \times n}$ is already inefficient

- Newton makes particularly sense, if the **Hessian is sparse**
  - Sparse Hessian $\leftrightarrow$ graphical models of dependencies



  - Factor graphs, large-scale structured least squares problems (cf. ceres)
  - in robotics: path optimization, computer vision: bundle adjustment, graph SLAM (cf. gtsam), probabilistic inference (MAP)

Approximate Newton Methods

- **Least Squares problems** and the Gauss-Newton approximation!
  - Very important problem class – ubiquitous in AI, ML, robotics, etc
  - Approximates the Hessian, scalable if the **Jacobian is sparse**

Approximate Newton Methods

- **Least Squares problems** and the Gauss-Newton approximation!
  - Very important problem class – ubiquitous in AI, ML, robotics, etc
  - Approximates the Hessian, scalable if the **Jacobian is sparse**

- Other methods approximate the Hessian from gradient observations:
  - BFGS, (L)BFGS (“quasi-Newton method”) – a default solver in science
  - Conjugate Gradient

Gauss-Newton method

- Consider a **least squares** problem (cost is a **sum-of-squares**):

$$\min_x f(x) \qquad \text{where } f(x) = \phi(x)^\top \phi(x) = \sum_{i=1}^d \phi_i(x)^2$$

with **features** $\phi(x) \in \mathbb{R}^d$, and we can evaluate $\phi(x)$ and $J = \frac{\partial}{\partial x}\phi(x)$ for any $x \in \mathbb{R}^n$

Gauss-Newton method

- Consider a **least squares** problem (cost is a **sum-of-squares**):

$$\min_x f(x) \qquad \text{where } f(x) = \phi(x)^\top \phi(x) = \sum_{i=1}^{d} \phi_i(x)^2$$

with **features** $\phi(x) \in \mathbb{R}^d$, and we can evaluate $\phi(x)$ and $J = \frac{\partial}{\partial x}\phi(x)$ for any $x \in \mathbb{R}^n$

- $\phi(x) \in \mathbb{R}^d$ is a vector; each entry contributes a squared cost term to $f(x)$
- $\frac{\partial}{\partial x}\phi(x)$ is the **Jacobian** $(d \times n$-matrix)

$$J = \frac{\partial}{\partial x}\phi(x) = \begin{pmatrix} \frac{\partial}{\partial_{x_1}}\phi_1(x) & \frac{\partial}{\partial_{x_2}}\phi_1(x) & \cdots & \frac{\partial}{\partial_{x_n}}\phi_1(x) \\ \frac{\partial}{\partial_{x_1}}\phi_2(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial}{\partial_{x_1}}\phi_d(x) & \cdots & \cdots & \frac{\partial}{\partial_{x_n}}\phi_d(x) \end{pmatrix} \in \mathbb{R}^{d \times n}$$

Gauss-Newton method

- The gradient and Hessian of $f(x)$ are

$$f(x) = \phi(x)^\top \phi(x)$$
$$\nabla f(x) = 2 \tfrac{\partial}{\partial x}\phi(x)^\top \phi(x) \qquad \text{(recall } \nabla f(x) \equiv \tfrac{\partial}{\partial x} f(x)^\top\text{)}$$
$$\nabla^2 f(x) = 2 \tfrac{\partial}{\partial x}\phi(x)^\top \tfrac{\partial}{\partial x}\phi(x) + 2\phi(x)^\top \nabla^2 \phi(x)$$

- *The Gauss-Newton method is the Newton method for $f(x) = \phi(x)^\top \phi(x)$ while approximating $\nabla^2 \phi(x) \approx 0$, i.e.*

$$\nabla^2 f(x) \approx 2 \tfrac{\partial}{\partial x}\phi(x)^\top \tfrac{\partial}{\partial x}\phi(x) = 2 J^\top J$$

(Use this approximation when computing the step δ is the standard Newton algorithm.)

Gauss-Newton method

- The approximate Hessian $H = 2J^\top J$ is **always semi-pos-def!**

Gauss-Newton method

- The approximate Hessian $H = 2J^\top J$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2 \sum_{i=1}^{d} \nabla\phi_i(x) \nabla\phi_i(x)^\top$$

  (which implies semi-pos-def)

Gauss-Newton method

- The approximate Hessian $H = 2J^\top J$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2 \sum_{i=1}^{d} \nabla\phi_i(x) \nabla\phi_i(x)^\top$$

  (which implies semi-pos-def)

- If the Jacobian J is sparse, so is the Hessian $\to$ graphical structure

Gauss-Newton method

- The approximate Hessian $H = 2J^\top J$ is **always semi-pos-def!**

- H is a sum of rank-1 matrices:

$$H = 2\sum_{i=1}^{d} \nabla\phi_i(x) \nabla\phi_i(x)^\top$$

  (which implies semi-pos-def)

- If the Jacobian J is sparse, so is the Hessian $\to$ graphical structure

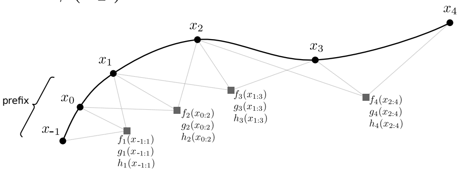
- H can be interpreted as pullback of the Euclidean norm $\phi^\top\phi$ in feature space. As it is x-dependent, this is a non-constant metric in x-space – it defines a *Riemannian* metric. (See math notes.)

Robotics example

- Path optimization: Let $x = (x_1, .., x_T), x_t \in \mathbb{R}^n$ be a discretized path,

$$\min_x \sum_{t=1}^{T} (x_t + x_{t\text{-}2} - 2x_{t\text{-}1})^2 \ + \ \phi(x_T)^2$$

where $x_0, x_{\text{-}1}$ are given, and $\phi(x_T)$ are some features of the end configuration x_T



Toussaint: *A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference*. 2017

- We use the formulation in terms of **features** throughout, also for hard constraints

Quasi-Newton methods

Quasi-Newton methods

- Assume we *cannot* evaluate $\nabla^2 f(x)$. *Can we still use 2nd order methods?*
- Yes: We can approximate $\nabla^2 f(x)$ from the data $\{(x_i, \nabla f(x_i))\}_{i=1}^k$ of previous iterations
- (General view: We can *learn* from the data $\{(x_i, \nabla f(x_i))\}_{i=1}^k$ $\rightsquigarrow$ e.g., Bayesian optimization or model-based optimization for blackbox optimization.)

Basic example

- We've seen two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$ – How can we estimate $\nabla^2 f(x)$?

Basic example

- We've seen two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$ – How can we estimate $\nabla^2 f(x)$?

- In 1D:

$$\nabla^2 f(x) \approx \frac{\nabla f(x_2) - \nabla f(x_1)}{x_2 - x_1}$$

Basic example

- We've seen two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$ – How can we estimate $\nabla^2 f(x)$?

- In $\mathbb{R}^n$: Let $y = \nabla f(x_2) - \nabla f(x_1)$, $\delta = x_2 - x_1$
  What are matrices H or $H^{\text{-}1}$ to fulfil the following?

  $$H\,\delta \stackrel{!}{=} y \qquad \text{or} \qquad \delta \stackrel{!}{=} H^{\text{-}1} y$$

  (The first equation is called *secant equation*)

Basic example

- We've seen two data points $(x_1, \nabla f(x_1))$ and $(x_2, \nabla f(x_2))$ – How can we estimate $\nabla^2 f(x)$?

- In $\mathbb{R}^n$: Let $y = \nabla f(x_2) - \nabla f(x_1)$, $\delta = x_2 - x_1$
  What are matrices H or $H^{\text{-}1}$ to fulfil the following?

$$H\,\delta \overset{!}{=} y \qquad \text{or} \qquad \delta \overset{!}{=} H^{\text{-}1} y$$

  (The first equation is called *secant equation*)

- "Simplest" symmetric rank-1 solutions for $\bar H \approx H$ and $\hat H \approx H^{\text{-}1}$:

$$\bar H = \frac{y y^\top}{y^\top \delta} \qquad \text{or} \qquad \hat H = \frac{\delta \delta^\top}{\delta^\top y} \tag{1}$$

  [Left: how to update $\bar H \approx H$. Right: how to update directly $\hat H \approx H^{\text{-}1}$.]

BFGS

- Broyden-Fletcher-Goldfarb-Shanno (BFGS) method:

Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x)$, tolerance θ
Output: x

```
initialize $\hat H = \mathbf{I}_n$
repeat
    compute $\delta = -\hat H \nabla f(x)$
    perform a line search $\min_\alpha f(x + \alpha\delta)$
    $\delta \leftarrow \alpha\delta$
    $y \leftarrow \nabla f(x+\delta) - \nabla f(x)$
    $x \leftarrow x + \delta$
    update $\hat H \leftarrow \left(\mathbf{I} - \frac{y\delta^\top}{\delta^\top y}\right)^\top \hat H \left(\mathbf{I} - \frac{y\delta^\top}{\delta^\top y}\right) + \frac{\delta\delta^\top}{\delta^\top y}$
until $\|\delta\|_\infty < \theta$
```

- The blue term is the $\hat H$-update as on the previous slide
- The red term "deletes" "old" $\hat H$-components. Check: $\hat H y = \delta$
- equivalent to the Sherman-Morrison formula: $H \leftarrow H - \frac{H\delta\delta^\top H^\top}{\delta^T H \delta} + \frac{yy^\top}{y^\top\delta}$

L-BFGS

- In high dimensions, we do not want to explicitly store a dense $\hat{H}$. Instead we store vectors $\{v_i\}$ such that $\hat{H} = \sum_i v_i v_i^\top$
- **L-BFGS** (limited memory BFGS) limits the rank of the $\hat{H}$ and thereby the used memory (number of vectors v_i)

L-BFGS

- In high dimensions, we do not want to explicitly store a dense $\hat{H}$. Instead we store vectors $\{v_i\}$ such that $\hat{H} = \sum_i v_i v_i^\top$
- **L-BFGS** (limited memory BFGS) limits the rank of the $\hat{H}$ and thereby the used memory (number of vectors v_i)

- Some thoughts:
  In principle, there are alternative ways to estimate $H^{\text{-}1}$ from the data $\{(x_i, f(x_i), \nabla f(x_i))\}_{i=1}^k$, e.g. using Gaussian Process regression with derivative observations
  - not only the derivatives but also the value $f(x_i)$ should give information on $H(x)$ for non-quadratic functions
  - should one weight 'local' data stronger than 'far away'?
    (GP covariance function)
  - related to model-based search (see Blackbox Optimization lecture)

(Nonlinear) Conjugate Gradient

Conjugate Gradient

- The “Conjugate Gradient Method” is a method for solving (large, or sparse) linear eqn. systems $Ax + b = 0$, without inverting or decomposing A. The steps will be “A-orthogonal” (=conjugate).
  We mention its extension for optimizing nonlinear functions $f(x)$

- As before we evaluted $g' = \nabla f(x_1)$ and $g = \nabla f(x_2)$ at points $x_1, x_2 \in \mathbb{R}^n$
- Additional assumption: *exact line-search* step to x_2:
  - $x_2 = x_1 + \alpha\delta_1 \ , \quad \alpha = \text{argmin}_\alpha \, f(x_1 + \alpha\delta_1)$
  - iso-lines of $f(x)$ at x_2 are tangential to δ_1

$\Rightarrow$ The next search direction should be “orthogonal” to the previous one, but orthogonal w.r.t. the Hessian H, i.e., $\delta_2^\top H \delta_1 = 0$, which is called conjugate

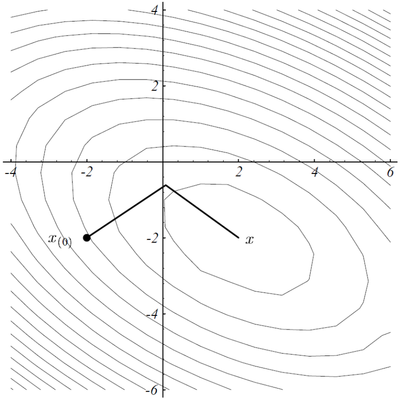
Conjugate Gradient

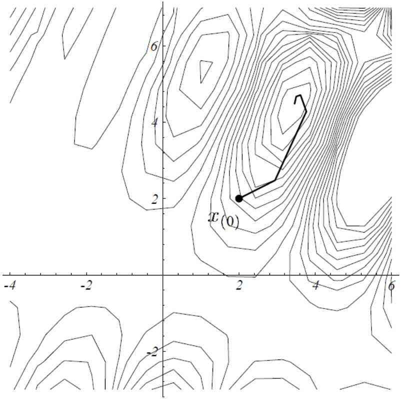
Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x)$, tolerance θ
Output: x

initialize descent direction $\delta = g = -\nabla f(x)$
repeat
  $\alpha \leftarrow \mathrm{argmin}_\alpha\, f(x + \alpha\delta)$ *// line search*
  $x \leftarrow x + \alpha\delta$
  $g' \leftarrow g,\ \ g = -\nabla f(x)$ *// store and compute grad*
  $\beta \leftarrow \max\left\{\frac{g^\top(g-g')}{g'^\top g'}, 0\right\}$
  $\delta \leftarrow g + \beta\delta$ *// conjugate descent direction*
until $|\Delta x| < \theta$

– $\beta > 0$: The new descent direction always adds a bit of the old direction!
– This *momentum* essentially provides 2nd order information
– The equation for β is by Polak-Ribière: On a quadratic function $f(x) = x^\top A x + b^\top x$ this leads to **conjugate** search directions, $\delta'^\top A\delta = 0$.

Conjugate Gradient





- For quadratic functions CG converges in n iterations.
  But each iteration does *exact* line search

Further Methods

- Beyond the standard canon – but perhaps discussed later:
  - Bound constrained optimization
  - Stochastic Gradient

  - Blackbox Optimization, Bayesian Optimization
  - model-based optimization, implicit filtering
  - Stochastic Search, Evolutionary Algorithms, EDAs
  - Simulated annealing
  - Nelder-Mead downhill simplex, pattern search
  - Rprop